I. Project prototype

1) Task description

Constant development of computing units and algorithms changed people's approach to data processing. Availability of this technical innovations leads to automatization of daily tasks in areas of industry and public life. There is a high demand for solutions that allow computers to solve tasks the way humans beings do, for example computer vision.

Computer vision is a branch of artificial intelligence focused on analysing scenes, objects and events on images and videos. It allows computers to extract information from images the same way people do. Application of neural networks (especially convolutional ones) resulted in many achievements in the area. Computer vision is broadly used in security cameras, biometrics, autonomous cars and other systems where surrounding should be analysed.

Machine learning and deep learning are commonly used to solve computer vision tasks. Machine learning is a type of data analysis method that, in contrast to regular methods, is capable to self-improve based on experience from the past. Term „machine learning" was introduced in 1959 by Arthur Samuel, with definition: „*Field of study that gives computers the ability to learn without being explicitly programmed"*. Concept of deep learning refers to broader family of machine learning algorithms based on artificial neural networks (ANNs). ANN is a type of mathematical structure that process information the same way biological nervous system does but with couple simplifications. Neural network consists of connected neurons arranged in layers. In computer vision problems convolutional neural networks (CNNs) are used because of their crucial capability to recognize patterns in images.

Human detection will be discussed as a part of the project. Human detection is a method of locating all human instances in given image or frame. The most basic implementation of human detection is expected to return exact position of all human beings present on the scene. More complex ones should also determine other parameters of each person like movement direction and speed. Newest techniques able to create pixel perfect masks that are pointing only at pixels related to individual instances present on the image or are able to estimate pose of detected instance based on created skeleton of person.

After performing human detection it is also possible to conduct identity recognition on detected person with facial recognition or pose estimation. This task is more difficult because it requires algorithm that possesses unique biometrical data of each individual person.

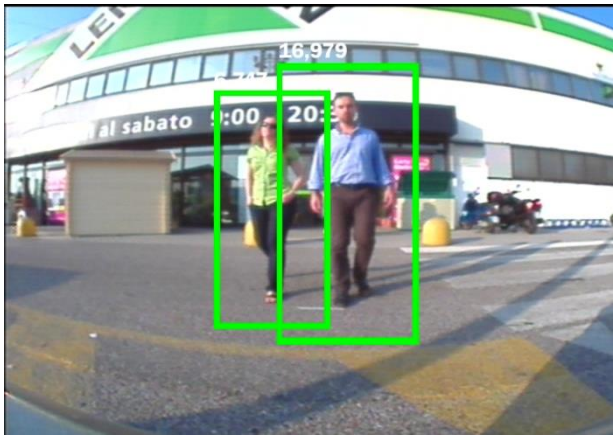

Figure 1 Example result of simple human detection task. Green boxes reveal current location of persons

Our main goal is to implement working human detection system. In order to achieve this we have to choose existing dataset and algorithm suitable for our needs. Dataset should be a large collection of images consisting of pedestrians in different sceneries. Quality of given dataset will influence our results. While choosing the correct dataset we should pay attention to collection size, scene diversity and type of output data. After choosing optimal training set, algorithm compatible with held dataset should be fitted. Reading scientific papers describing existing human detection systems might be helpful in decision-making.

Having both dataset and algorithm we can get on with implementation. During this process we will run algorithm with different parameters values in order to determine which configuration is best for our problem.

Human detection methods are commonly used in video surveillance systems. After detection they are able to determine how many persons are present on the scene or whether people are acting in a safe manner. For example human detection system operating on subway station could spot people who fell on the track. Also, autonomous cars systems could not operate without precise human detection system that can determine if pedestrians are on a collision course or are trying to trespass the road. There is also a threat that this system in the wrong hands could be used to abuse people's freedom, for example by authorities to control crowds during protests.

2) Algorithm description

While choosing an algorithm, which fits our needs we considered many different approaches. Starting from a few computer vision examples we went through many techniques that employ neural networks as the base. After exact analysis, we decided that mask RCNN is the one which suits our task and dataset the best. To explain our decision we will describe every algorithm shortly and explain our decision.

The first one is the histogram of oriented gradients (HOG)[1][2]. The essential thought is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

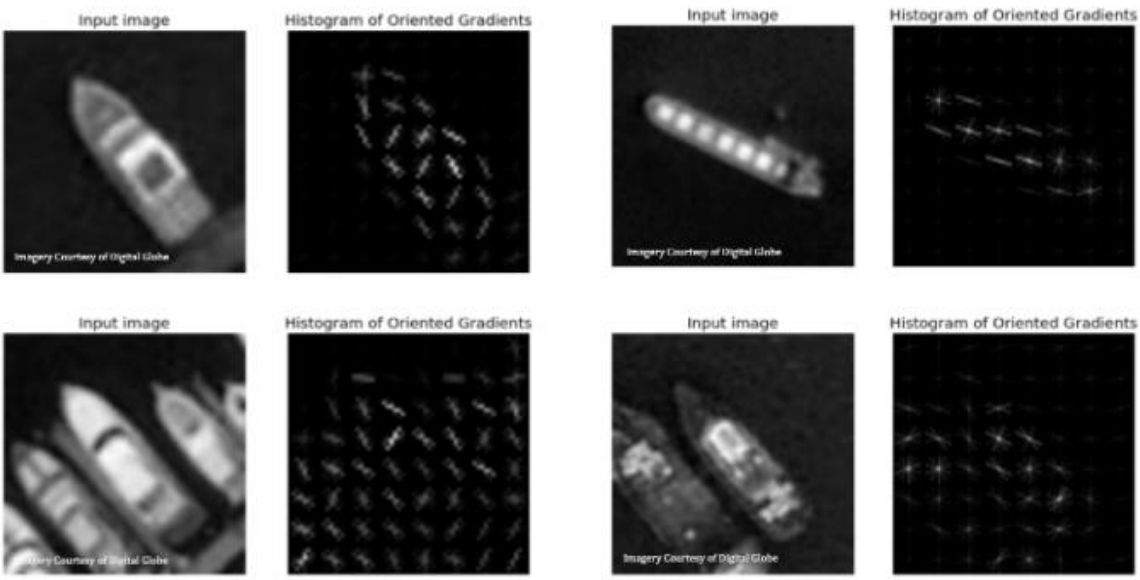

Figure 2 Sample satellite image cutouts of boat(s) (left), and image of HOG descriptor (right) (Imagery Courtesy of Digital Globe)

The HOG**[3]** descriptor has a few key advantages over other descriptors. Since it operates on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions. Moreover, as Dalal and Triggs discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permit the individual body movement of pedestrians to be ignored so long as they maintain a roughly upright position. Despite that, HOG-based classifiers lack the power of neural network classifiers, and so may break down in crowded or complex scenes.

The second computer vision algorithm which we analyzed was the **Viola-Jones approach**, which is mostly used for face detection. The basic idea is to calculate the sum of pixels within rectangular areas of the picture. The picture illustrates four different types of features that are applied in that framework.

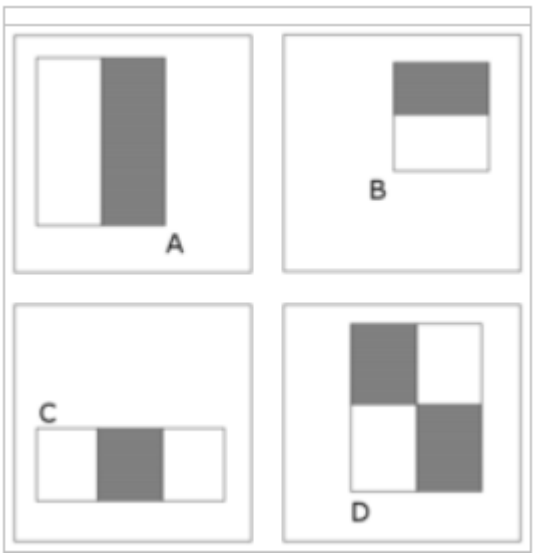

Figure 3 Example rectangle features shown relative to the enclosing detection window

The value of the feature equals the sum of the pixel within clear rectangles extracted from pixels sums from outside of the shaded rectangles. Their feedback is considerably coarse because they are sensitive

to vertical and horizontal features. The advantages of this approach are very high detection rates and real-time processing. However, it can be used only for face detection[3].
According to the moderns approaches we decide to consider some algorithms which use neural networks. The first and most simple one was **CNN**.

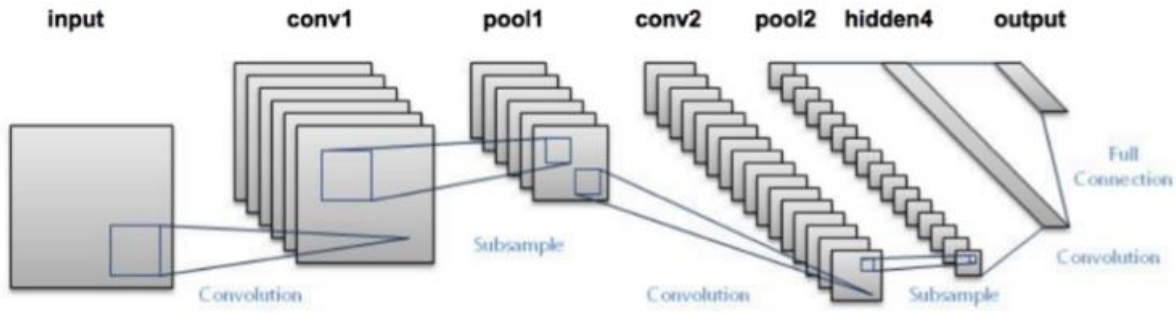

Figure 4 CNN architecture

It passes an image through a network similar to the one showed in the picture. After going through many convolutions and pooling layers we get the picture's class as an output.

To use this method to object detection problem we first take an image as an input. Then we divide it into multiple regions and consider each region as a separate image. Then we pass all the regions into CNN to group them into multiple classes. After all of these steps, we can finally combine all the regions into one image with detected objects.

The problem with using this approach is that in some cases the image might be covering all the images, while in others it can take only a small part. We should also expect that one object can be visualized in many different shapes. Those factors require producing a huge amount of regions, which calculations take a huge amount of time. To solve this issue we can use region-based CNN[4]**.**

This approach divides the image into regions using a selective search method and then checks if it contains an object. It relives an algorithm from a duty of dividing a picture to many small pieces. Instead, it creates only as many regions as it needs.

Four regions form an object: varying scales, colors, textures and enclosure. The selective search identifies these patterns in the image and based on that, proposes various regions. It always follows a few steps. Firstly, it takes an image as an input. Then, it generates initial sub-segmentations so that we have multiple regions from this image. The technique then combines similar regions to form a larger region (based on color similarity, texture similarity, size similarity, and shape compatibility). Finally, these regions then produce the final object locations (Region of Interest).

Figure 5Fisualization of the algortihm

The whole RCNN process can be described in a few steps:

1. Take the pre-trained convolutional network.

2. Retrain model – train the last layer basing on the number of classes that have to be detected.

3. Get regions.

4. Reshape regions that they can fit CNN input size.

5. Train the binary SVM model to differentiate the background from an object.

6. Train the linear regression model to generate tighter bounding boxes for each object in the image.

The drawback of this approach is the computation time which is around 40-50 seconds for each image and the fact that the entire process of object detection uses three models. To fix those issues we need to change this solution a little bit.

An algorithm that fixes our problem is called **fast RCNN.** Instead of running a CNN many times per image, we can run it just once and get all the regions containing some object.

We can do it by sharing one computation across many regions. In Fast RCNN, we feed the input image to the CNN, which in turn generates the convolutional feature maps. Using these maps, the regions of proposals are extracted. We then use an RoI pooling layer to reshape all the proposed regions into a fixed size, so that it can be fed into a fully connected network.

Summarizing this process into a few steps we need to:
1. Pass the image to a ConvNet which in turn generates the Regions of Interest.
2. Apply the RoI pooling layer on all of these regions to reshape them and fix the input of ConvNet.
3. Pass each region to a fully connected network.
4a. Use the softmax layer on top of the fully connected network to output classes.
4b. Along with the softmax layer, a linear regression layer is also used parallel to output bounding box coordinates for predicted classes.

This is how Fast RCNN resolves two major issues of RCNN, i.e., passing one instead of many regions per image to the ConvNet, and using one instead of three different models for extracting features, classification and generating bounding boxes.

Solving this problem the only feature which we wanted to add to our output was a mask that will distinguish found objects from the rest of the picture. In this case, we needed to use a modified version of fast RCNN called mask RCNN.

Mask R-CNN[5], extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression. The mask branch is a small Fully Convolutional Network applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Mask R-CNN is simple to implement and train given the Fast R-CNN framework, which facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation.

Mask R-CNN surpasses all previous state-of-the-art single-model results on the COCO instance segmentation task, including the heavily engineered entries from the 2016 competition winner. As a by-product, this method also excels on the COCO object detection task.

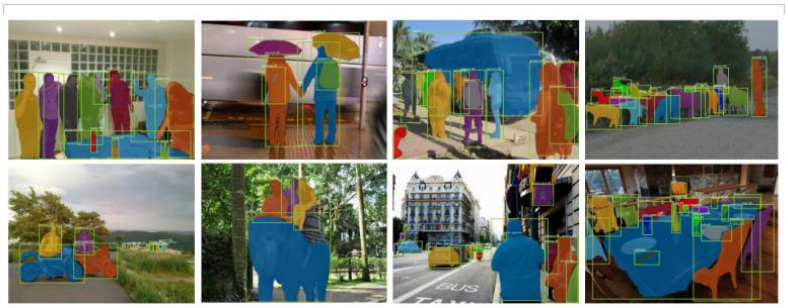

Figure 6 Mask R-CNN results on the COCO test set. these results are based on ResNet-101, acheving a mask AP of 35.7 and running at 5 fps.. Mask are shown in color.

3) Testing images/recordings selection

We considered two main methods of collecting images' dataset: making and labelling own images or using already available dataset.
The advantages of first option are:

- our personal impact on the shape of the dataset
- possibility to fit dataset to the future environment of usage

On the other side, using existing dataset has many important pluses. Among the most important there should be mentioned:

- diverse surroundings
- saving time on collecting data and labelling it
- cross-environmental conditions like brightness, overlapping objects ect.
- eliminating highly likely errors in collected data (popular datasets were alrealy tested by thousands users)

We took into account above considerations and high availability of databases with humans labelled [6] [7] [8]. Finally we decided to choose existing dataset.

There are many available datasets for human detections. The article with the biggest amount of citations in field of Human Detection which introduce HOG [9] uses MIT
pedestrian database . It is worth noting that even authors of this article mention that this set is not complicated. They mention also INRIA dataset which contain more examples.
Popular dataset in field of image processing with humans is MPII [8]. Authors of it are however more focused on pose estimation. That's not a topic of the project.

Another noteworthy dataset for object detection is COCO. Advantages of it are:

- the dataset is still developed
- COCO dataset can be use in commercial uses. That's really important in developing commercial solutions
- It contains over 330 thousands of data samples. There are 66808 photos with tagged humans. It is the biggest amount
- there are often more than one person on the picture
- contains people in different environments and in different positions
- very cross-sectional
- takes account of different conditions like:
    - mutual covering of people
    - different positions to the camera
    - more than one person on the picture

Figure 7 Sample 1 from COCO

Figure 8 Sample 2 from COCO

Figure 9 Sample 3 from COCO

Dataset	MIT Pedestrain	MPII Human Pose	INRIA	COCO
#training	509	28821	1805	64115
#test	200	11,701	-	2693
Licence	Only for research	MIT	MIT	MIT

The advantage of COCO dataset is also available API. This tool let us simply load and aggregate needed part of the dataset.
Taking into account amount of classes in this dataset, ability to efficiently use it would be a great prevalence in another image recognition project for authors of this document.

COCO dataset has thoughtful structure: it contains images and annotations:

Images	Annotations
2017 Train images [118K/18GB]	2017 Train/Val annotations [241MB]
2017 Val images [5K/1GB]	
2017 Test images [41K/6GB]	

In annotations there should be distinguished following data:

- instances – locations of classes on photos

Figure 10 Location of classes on photos in COCO

- person_keypoints – keypoints of human pose

Figure 11Location of keypoints in COCO

- captions – descriptions of photos

```
A woman wearing a dress and holding an umbrella.
The woman in the pink skirt is holding an umbrella.
The woman is walking carefully through the leaves.
A woman in a dress has a handbag and is holding an umbrella.
A young female in a pink skirt is holding an umbrella.
```

In the project there would be mainly used instances.

We used existing and expanded by the python community API *cocoAPI* [10]. Part of the developers' team used WINDOWS operating system. In this case it is worth to mention usefulness of accompanying repository *Clone of COCO API* [11].

There are several important functions and classes of this API:

```
class COCO:

COCO helper class for reading and visualizing annotations.

        fun loadCats: Load cats with the specified ids

        fun getImgIds: Get img ids that satisfy given filter conditions

        fun loadImgs: Load anns with the specified ids

        fun showAnns: Display the specified annotations.
```

```
class COCOeval:
Initialize CocoEval using coco APIs for gt and dt


… and others. More is shown in demo jupyter notebooks in repository.
```

[1] “Histogram of Oriented Gradients (HOG) Boat Heading Classification.” [Online]. Available: https://medium.com/the-downlinq/histogram-of-oriented-gradients-hog-heading-classification-a92d1cf5b3cc. [Accessed: 20-Nov-2019].
[2] “Real-time Human Detection in Computer Vision — Part 1.” [Online]. Available: https://medium.com/@madhawavidanapathirana/https-medium-com-madhawavidanapathirana-real-time-human-detection-in-computer-vision-part-1-2acb851f4e55. [Accessed: 20-Nov-2019].
[3] “Deep Learning Haar Cascade Explained - Will Berger.” [Online]. Available: http://www.willberger.org/cascade-haar-explained/. [Accessed: 20-Nov-2019].
[4] “A Step-by-Step Introduction to the Basic Object Detection Algorithms (Part 1).” [Online]. Available: https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/. [Accessed: 20-Nov-2019].
[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN.”
[6] “INRIA Person dataset.” [Online]. Available: http://pascal.inrialpes.fr/data/human/. [Accessed: 24-Oct-2019].
[7] “COCO - Common Objects in Context.” [Online]. Available: http://cocodataset.org/#home. [Accessed: 24-Oct-2019].
[8] “MPII Human Pose Database.” [Online]. Available: http://human-pose.mpi-inf.mpg.de/. [Accessed: 24-Oct-2019].
[9] B. T. Navneet Dalal, “Histograms of Oriented Gradients for Human Detection,” 2005, pp. 886–893.
[10] “cocoapi/PythonAPI at master · cocodataset/cocoapi.” [Online]. Available: https://github.com/cocodataset/cocoapi/tree/master/PythonAPI. [Accessed: 24-Oct-2019].
[11] “philferriere/cocoapi: Clone of COCO API - Dataset @ http://cocodataset.org/ - with changes to support Windows build and python3.” [Online]. Available: https://github.com/philferriere/cocoapi. [Accessed: 24-Oct-2019].